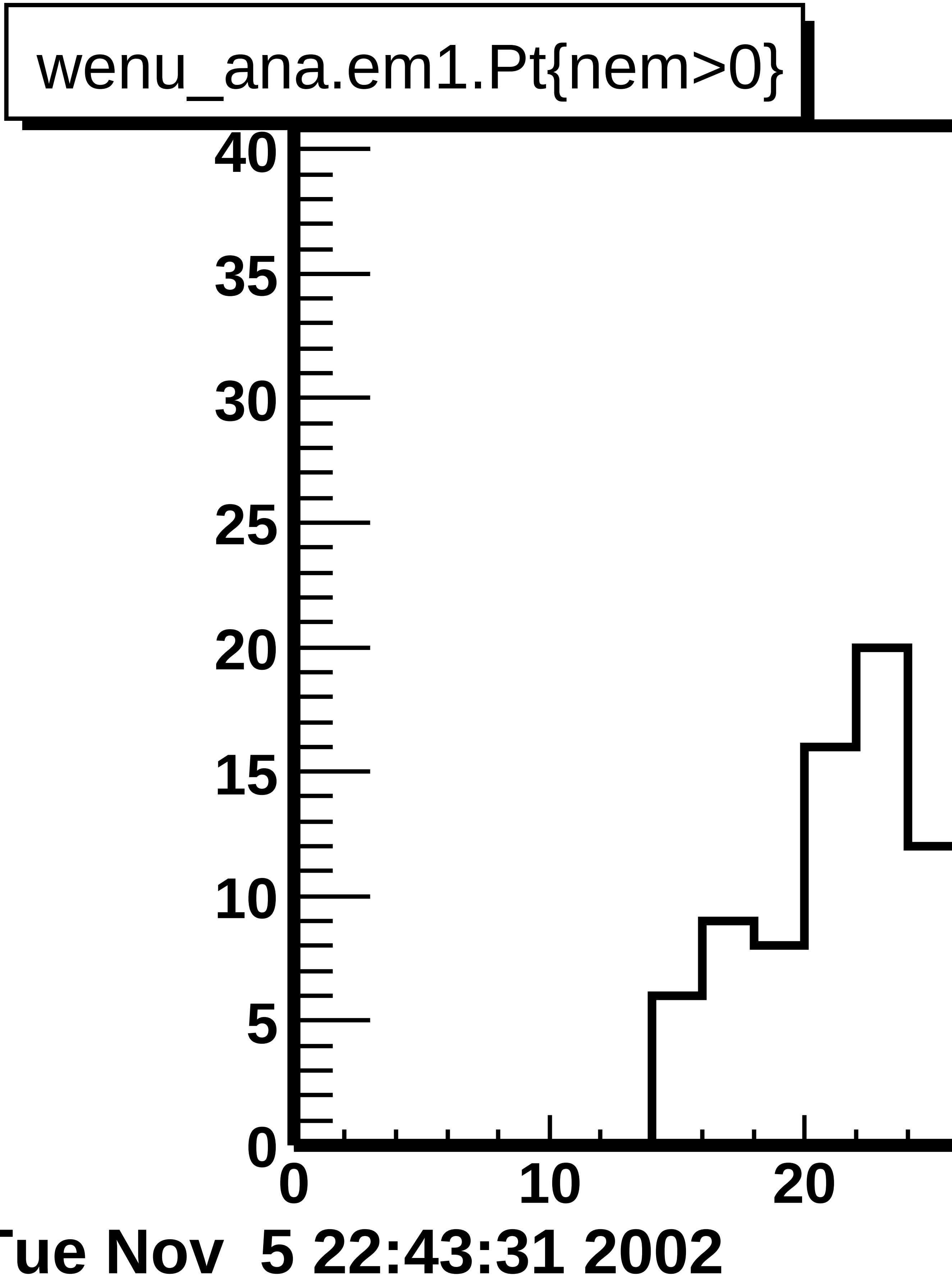
The ‘truffle’ analysis package

- An object-oriented system for root analysis.
- Originally based on Ariel’s framework.
  - But by now completely different in implementation.
  - Much more powerful and flexible.
- Goals:
  - Convenience for interactive use.
  - Can add new object types and new data sources without changing the core.
- Can presently read tmb_tree and recoanalyze tuples.
  - Reading thumbnails directly would be possible too, but has not been implemented due to speed and code size issues.

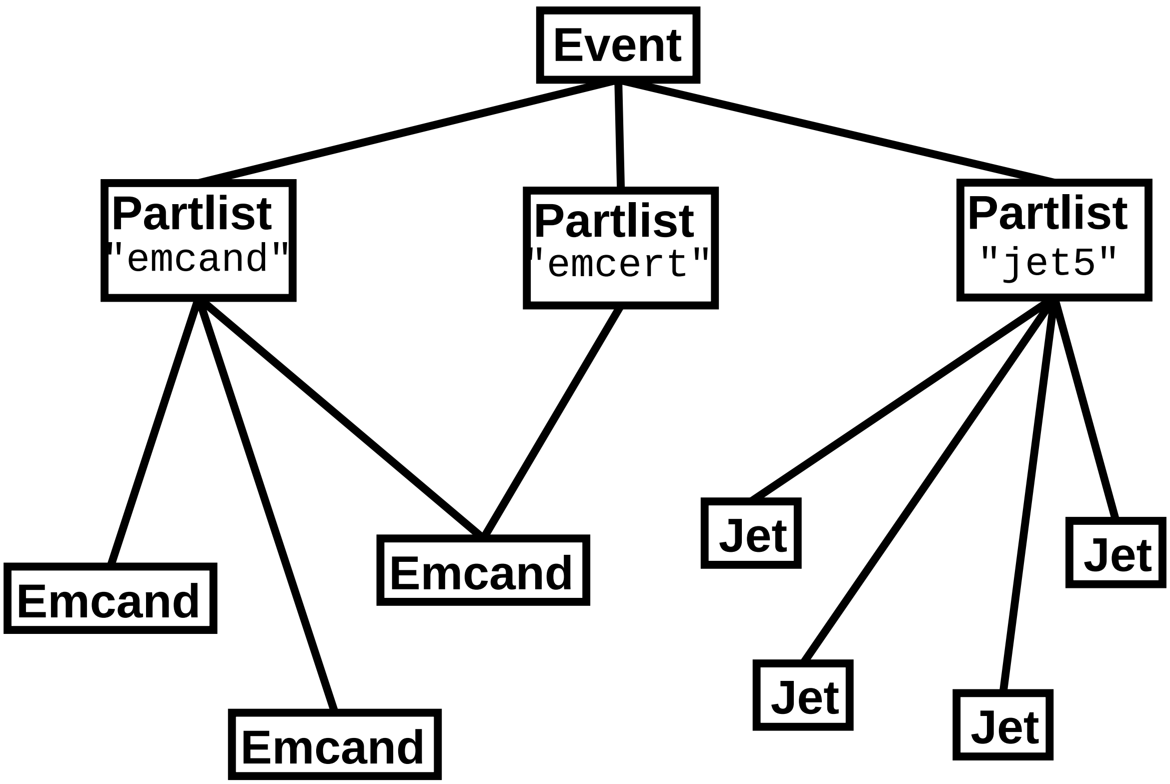
roo
roo
tru
tru
tru
tru
tru
tru
tru
Inf
tru

•



Event Model

- A collection of named lists of particles.
- Particles derive from a common `Particle` class.
  - Which in turn derives from `Lorentz_Vector`.
  - Particles are reference-counted:
    - A single instance may be in multiple lists.



- Particle lists are not built until referenced.
- Lists are built by "builder" objects. `Builder`.
- `Builder` instances are registered with `Event`.
- Encapsulates particle-id, selection, transformation.

Retrieving data from `Event`

```
Event ev = ...;

// Number of particles.
int n = ev.npart ("emcand");

// List of particles.
const Partlist& parts = ev.parts ("emcand");

// Individual particles.
// (Index is 1-based.)
const Particle& part = ev.part (1, "emcand");
```

- Also have convenience methods for various particle types.

```
// Get EM candidates.
// List name defaults to "emcert".
int n = nem();
const Empart& empart = ev.em (1);
const Empart& emcand = ev.em (1, "emcand");
```

Derived `Particle` classes

- `Cps` — A CPS cluster.
- `Emcand` — Results from electron certification.
- `Emclust` — Raw EM cluster object.
- `Fps` — A FPS cluster.
- `Iso_Track` — An isolated charged track.
- `Jet` — Result of jet finding; either using the calorimeter, MC particles, or MC partons. Either corrected or not.
- `Mcinfo` — MC cross section and process information.
- `Mcpart` — A MC particle.
- `Mcvtx` — A MC decay vertex.
- `Met` — Missing E_T, either corrected or not.
- `Mucand` — Results from muon certification.
- `Track` — A charged track.
- `Trig` — Which trigger bits fired.
- `Unclustered` — Unclustered calorimeter energy.
- `Vertex` — A tracking vertex (either primary or secondary).

Plotting features

- The framework supplies a convenient shorthand for making plots interactively.
- To access these features interactively, call the `shell()` function. Any input that the framework doesn't recognize will be passed on to root. Use "`.qt`" to exit.
- No time to describe this in detail, so just present some examples.
- `d wjets[10:15].em1.Pt nem>0 50 0 100`
  Plot the p_T of the leading certified electron in the 'wjets' sample. Use only events 10–15, and plot only those that have at least one electron. Use 50 bins, in range 0–100.
- `d wjets.em=emtightt;em1.Pt nem>0 50 0 100 SAME`
  The same, but scan all events and use '`emtightt`' electrons (tight cut with track match). Plot the result on the same canvas as the previous plot.
- `` d wjets.em1.DeltaR(mcem1`mcwem) nem>0 50 0 1 ``
  Arbitrary CINT expressions may be used. Plot ΔR between the leading reconstructed electron and the MC electron from W decay.

More plotting examples

- `d wjets.jet$i.Pt`
  Plot the p_T of all jets in the event. The '`$i`' makes an implicit loop over all jet in the event.

- `d wjets.(jet1+jet2).M() njet>=2`
  Plot the invariant mass of the leading two jets.

- `d wjets.(jet$i+jet$j).M() $i>$j`
  Plot the invariant masses of all dijet combinations.

- `d wjets.jet$i.Eta:jet$i.Phi ! 50 -3.5 3.5 50 -2 2`
  Plot the η—ϕ distribution of jets as a 2-D histogram. Unlike the standard root 2-D histograms, every point on the plot directly corresponds to an entry.

- `def goodev nem>0&&em1.Pt>20`
  `d wjets.em1.Pt $goodev`
  Simple macro definition.

- `def minv(a,b) ((a)+(b)).M()`
  `d wjets.$minv(jet1,jet2) njet>=2`
  Parameterized macro definition.

Some other features

- Use the **scan** command to print out a small number of variables. Conventions are the same as before.

```
> scan ttbar_ana[10:15].jet1.Pt:jet2.Pt njet>=2
    10   |  83.31537 | 45.755596
    11   |  74.64558 | 71.041191
    12   | 169.94582 | 102.21555
    13   | 45.338705 | 27.008058
    14   | 60.697147 | 26.020169
    15   | 79.842447 | 44.893623
```

- **zone** *nx ny* — Divide canvas into $nx \times ny$ pads. Draw commands will automatically step through the pads. Middle-click in a pad to draw in that one next.

- **print** *file* — Write the current canvas as encapsulated postscript to file.

Status

- Code is in the `truffle` CVS library, but at this time is not part of the weekly builds.
  - (Code has dependencies on other packages, like `emcandidate`, which are also not part of the weekly builds.)
- A pre-built version is available on clued0.
- See `http://www-d0.fnal.gov/~snyder/truffle.html` for instructions.

Future directions

- Work on efficiency issues.
- Move some processing to the tmb_tree generation step.
- Write some documentation.
- Make truffle code callable from python.
  - Have code to use cint to semi-automatically generate boost.python wrappers from C++ headers. I've used this to wrap CLHEP classes for python; I'd also like to expand this to allow writing d0 framework packages in python. Needs more work, though.